Completeness Guarantees for Incomplete Reasoners

Giorgos Stoilos, Bernardo Cuenca Grau, and Ian Horrocks

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, UK

Abstract. We extend our recent work on evaluating incomplete reasoners by introducing *strict testing bases*. We show how they can be used in practice to identify ontologies and queries where applications can exploit highly scalable incomplete query answering systems while enjoying completeness guarantees normally available only when using computationally intensive reasoning systems.

1 Introduction

A key application of OWL ontologies is ontology-based data access [12, 9, 3, 2, 7, 11], where an ontology is used to support query answering against distributed and/or heterogeneous data sources. The ontology provides the vocabulary used to formulate queries, and a conceptual model (or schema) that is used in computing query answers. In a Semantic Web setting, a typical scenario would involve the use of an OWL ontology to answer SPARQL queries over RDF datasets.

Unfortunately, when using an expressive ontology language such as OWL, computing query answers can be very costly, and in a (Semantic) Web setting, datasets may be extremely large. There has therefore been a growing interest in the development of query answering systems that are highly scalable in practice, but that are not guaranteed to be *complete* in all cases; i.e., for some combinations of query, ontology and dataset, they will not compute all query answers. Most such systems (e.g., Oracle's Semantic Data Store, Sesame, Jena, HAWK, OWLim, Minerva, and Virtuoso) are based on database or RDF triple store technologies; others are based on approximate reasoning techniques [10, 5].

Although the scalability of such systems is attractive, application developers face two main difficulties when using them. Firstly, incomplete query answers may not be acceptable in a given application; and secondly, even if some incompleteness is acceptable, it may be important to know just how incomplete answers are likely to be, and to compare the scalability-completeness trade-off offered by different systems. One way to address these issues is via empirical testing, e.g., checking the answers given by query answering systems w.r.t. a particular ontology, dataset and query, and although primarily intended for performance testing, benchmark suites such as LUBM [4] have sometimes been used for this purpose. However, this kind of testing has serious limitations: results are specific to a given query, ontology and dataset, and may tell us nothing about

the behaviour of the system more generally; and, in order to determine the system's degree of completeness, we need to already know, or be able to compute, exact answers to the given queries.

In our recent work [13], we addressed these issues by introducing the notion of a *Testing Base* (TB). For a given ontology and query, a TB is a set of datasets such that, for any "well behaved" query answering system, if the system is complete for each dataset in the TB, then it will be complete for *any* dataset. As well as providing a quantitative measure of completeness, which we call the *completeness degree*, TBs thus allow us to identify circumstances under which a completeness guarantee can be provided even when the system being used is incomplete in general. This is very useful in practice given that in many applications the ontology and (kinds of) query are fixed at design time, or change relatively infrequently, whereas the data is typically unknown and/or frequently changing. Unfortunately, we were unable to devise a practical algorithm for computing TBs; instead, we devised an algorithm that efficiently computes an approximation of a TB. This algorithm can be used to approximate the completeness degree, but it cannot be used to provide completeness guarantees.

In this paper we extend our previous work in several directions. Most importantly, we define the notion of a *Strict Testing Base*; we show that strict TBs are typically much smaller than TBs, prove that they can be used to provide the same completeness guarantee as TBs, and present an efficient algorithm for computing them. This algorithm can thus be used to identify ontologies and queries where applications can exploit highly scalable incomplete systems while enjoying completeness guarantees normally available only when using computationally intensive reasoning systems—i.e., they can have *the best of both worlds*.

Additionally, we propose four properties that any "reasonable" measure of completeness should ideally enjoy, and we show that while completeness degree w.r.t. a TB satisfies all of these properties, completeness degree w.r.t. a strict TB satisfies only two of them, albeit the most important two.

Finally, our preliminary evaluation, which includes the LUBM ontology and queries as well as (a version of) the Galen ontology of clinical terms, suggests not only that strict TBs are easy to compute in practice, but also that completeness guarantees can often be provided for realistic ontologies and queries.

2 Preliminaries

Description Logics We assume that the reader is familiar with the basics of DL syntax, semantics and standard reasoning problems [1], and we use standard notions of a TBox $\mathcal{T}$ (the terminology, or conceptual schema) and an ABox $\mathcal{A}$ (the assertions, or data). In the context of ontology-based data access, the ontology may be thought of as consisting only of a TBox, with the data being stored in the sources. From an OWL point of view, however, we can treat the contents of the sources as ABox assertions, and the ontology as being the union of the TBox and ABox, i.e., $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$. To avoid conflating the schema (TBox) and the data (ABox), we consider only fragments of the DLs underpinning OWL

DL and OWL 2 that do not provide for *nominals* (i.e., that do not allow ABox individuals to be used to define TBox concepts); we also assume (without loss of generality) that ABoxes contain only *atomic assertions*—that is, each assertion of the form $C(a)$ or $R(a, b)$ in $\mathcal{A}$ must be such that C and R are atomic.

Queries We use the standard notions of term, (function-free) atom and variable. A datalog clause is an expression $H \leftarrow B_1 \wedge \ldots \wedge B_n$ where H (the *head*) is a (possibly empty) atom, $B_1 \wedge \ldots \wedge B_n$ (the *body*) is a conjunction of atoms, and each variable in the head also occurs in the body. A union of conjunctive queries (UCQ) is a tuple $u = \langle Q_P, P \rangle$ with Q_P a query predicate and P a finite set of datalog clauses such that Q_P is the only predicate occurring in head position in P and the body of each clause in P does not contain Q_P. We denote with $\mathsf{var}(q)$ the set of variables in q and say that a variable is distinguished if it appears in the head. Finally, q is a conjunctive query (CQ) if it is a UCQ and P has one clause. If $q = \langle Q_P, P \rangle$ is a CQ, we often abuse notation and write $q = P$; if u is a UCQ with $P = \{P_1, \ldots, P_n\}$, we write $u = \{q_1, \ldots, q_n\}$ with $q_i = P_i$ a CQ.

A tuple of constants $\vec{a}$ is a certain answer of a UCQ $q = \langle Q_P, P \rangle$ with respect to $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ iff $\mathcal{O} \cup P \models Q_P(\vec{a})$, where P is seen as a set of universally quantified implications with first-order semantics. The set of certain answers of q w.r.t. $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ is (equivalently) denoted as either $\mathsf{cert}(q, \mathcal{T}, \mathcal{A})$ or $\mathsf{cert}(q, \mathcal{O})$. Clearly, the set of certain answers satisfies the following useful properties:

1. *Monotonicity*: $\mathsf{cert}(q, \mathcal{O}) \subseteq \mathsf{cert}(q, \mathcal{O}')$ for each $\mathcal{O}$, $\mathcal{O}'$ and q with $\mathcal{O} \subseteq \mathcal{O}'$.
2. *Invariance under isomorphisms*: For each pair of *isomorphic ABoxes* $\mathcal{A}$ and $\mathcal{A}'$ (i.e., identical modulo renaming of individuals), $\mathsf{cert}(q, \mathcal{T}, \mathcal{A})$ and $\mathsf{cert}(q, \mathcal{T}, A')$ are also identical modulo the same renaming.

UCQ Rewritings Intuitively, a *UCQ rewriting* for a TBox $\mathcal{T}$ and a CQ q is a UCQ that extends q with the information from $\mathcal{T}$ that is relevant to answering the query. Formally, a *UCQ rewriting* for $\mathcal{T}$ and q is a UCQ u such that, for each ABox $\mathcal{A}$ where $\mathcal{T} \cup \mathcal{A}$ is consistent, the following properties hold:

1. (Soundness:) For each $q' \in u$, we have that $\mathsf{cert}(q', \emptyset, \mathcal{A}) \subseteq \mathsf{cert}(q, \mathcal{T}, \mathcal{A})$.
2. (Completeness:) $\mathsf{cert}(q, \mathcal{T}, \mathcal{A}) \subseteq \bigcup_{q' \in u} \mathsf{cert}(q', \emptyset, \mathcal{A})$.

Several well-known techniques can be used to reduce the size of (U)CQs. A CQ q is reduceable if it contains distinct body atoms that are unifiable. A reduction q' of q is obtained by applying the most general unifier θ to the body of q. A *condensation reduction* $\mathsf{cond}(u)$ is a UCQ obtained from u by ensuring that no two queries q, q' exist such that q' subsumes q and q' is a reduction of q. Finally, a *subsumption reduction* $\mathsf{sub}(u)$ is a UCQ obtained from u by ensuring that no two queries q, q' in the reduction are such that q' subsumes q.

Justifications Finally, our framework in [13] relies on the well-established notion of a *justification* for an entailment (see e.g., [6]). In the case of CQ answering, a justification for a CQ q and a tuple $\vec{a} \in \mathsf{cert}(q, \mathcal{O})$ in a consistent ontology $\mathcal{O}$ is an ontology $J \subseteq \mathcal{O}$ such that $\vec{a} \in \mathsf{cert}(q, J)$ and $\vec{a} \notin \mathsf{cert}(q, J')$ for each $J' \subset J$.

3 A Framework for Evaluating Completeness

In this section we present our revised and extended framework for evaluating the completeness of Semantic Web CQ answering systems.

3.1 CQ Answering Algorithms

Our framework adopts a rather general notion of a CQ answering algorithm. This allows us to abstract from the specifics of implemented systems and establish general results that hold for any system satisfying certain basic properties.

Definition 1. *A* CQ answering algorithm ans *for a DL* $\mathcal{L}$ *is a procedure that, for each* $\mathcal{L}$*-ontology* $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ *and CQ* $q = \langle Q_P, P \rangle$ *computes in a finite number of steps a set* $\mathsf{ans}(q, \mathcal{O})$ *of tuples of constants of the same arity as* Q_P*.*

- *It is* sound *if* $\mathsf{ans}(q, \mathcal{O}) \subseteq \mathsf{cert}(q, \mathcal{O})$ *for each* $\mathcal{O}$ *and* q*.*
- *It is* complete *if* $\mathsf{cert}(q, \mathcal{O}) \subseteq \mathsf{ans}(q, \mathcal{O})$ *for each* $\mathcal{O}$ *and* q*.*
- *It is* faithful *if it satisfies the same monotonicity and invariance under isomorphisms properties as* cert*.*
- *It is* compact *if for each consistent* $\mathcal{O}$*, each* q*, and each* $\vec{a} \in \mathsf{cert}(q, \mathcal{O}) \cap \mathsf{ans}(q, \mathcal{O})$*, there exists a justification* J *for* $q, \vec{a}$ *in* $\mathcal{O}$ *such that* $\vec{a} \in \mathsf{ans}(q, J)$*.*

Intuitively, ans is faithful if it implements the semantics of CQ answering in a "reasonable" way; in particular, the set of computed query answers for a fixed query can only grow if new axioms are added to the ontology (*monotonicity*) and the algorithm should be robust under trivial isomorphic renamings of individuals in the ABox (*invariance under isomorphisms*). Most of the results in our framework require ans to be at least *sound* and *faithful*, which we believe to be reasonable requirements that are satisfied by most if not all existing incomplete reasoners. For some of our results, however, *compactness* is also an issue. Intuitively, ans is compact if, whenever it correctly computes a certain answer $\vec{a}$ for some query q and ontology $\mathcal{O}$, then it will also compute $\vec{a}$ for q and some minimal subset of $\mathcal{O}$ that is sufficient to derive $\vec{a}$.

Consider an (incomplete) algorithm ans that, given $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ and q, ignores $\mathcal{T}$ and answers q only w.r.t. $\mathcal{A}$. Clearly, ans is sound and faithful. Furthermore, it is compact since, for each consistent $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ and certain answer $\vec{a} \in \mathsf{ans}(q, \mathcal{O})$, there is a minimal subset $\mathcal{A}'$ of $\mathcal{A}$ (a justification) that is sufficient to derive $\vec{a}$.[1] Suppose, however, that in order to handle atomic implications of the form $A \sqsubseteq B$, ans is extended as follows: it selects from $\mathcal{T}$ the set $\mathcal{T}'$ of atomic implications, extends $\mathcal{A}$ to $\mathcal{A}'$ by adding assertions implied by $\mathcal{T}'$ (e.g., adding $B(a)$ if $A(a) \in \mathcal{A}'$ and $A \sqsubseteq B \in \mathcal{T}'$), and uses $\mathcal{A}'$ to answer queries as before. Assume, however, that ans contains a bug, and only adds $B(a)$ if both $A(a)$ and $C(a)$ occur in $\mathcal{A}'$, for $C \neq A$ a (fixed) atomic concept. Despite the bug, the algorithm is still sound and faithful, but it is not compact. To see this, consider $\mathcal{T} = \{A \sqsubseteq B\}$ and q asking for the instances of B. For $\mathcal{A} = \{A(a), C(a)\}$ we have

[1] Recall that we are assuming that TBoxes do not contain nominals.

that $\mathsf{cert}(q, \mathcal{T}, \mathcal{A}) = \mathsf{ans}(q, \mathcal{T}, \mathcal{A}) = \{a\}$. However, the only relevant justification is $J = \mathcal{T} \cup \mathcal{A}_J$ for $A_J = \{A(a)\}$; but $a \notin \mathsf{ans}(q, J)$, and thus ans is not compact.

We believe that compactness is also a reasonable property to expect from a CQ answering algorithm, and that non-compactness is likely to be indicative of some "oddity" in the algorithm, as in the above example.

3.2 Testing Bases

Next, we briefly recapitulate from [13] the central notion of a *testing base*: a collection of minimal ABoxes (called *testing units*) which can produce an answer to q w.r.t. some minimal subset of $\mathcal{T}$. To check completeness, a testing base must include all "relevant" testing units.

Definition 2. *An ABox $\mathcal{A}$ is a* testing unit *for a CQ q and TBox $\mathcal{T}$ if $\mathcal{T} \cup \mathcal{A}$ is consistent and there exists a tuple $\vec{a} \in \mathsf{cert}(q, \mathcal{T}, \mathcal{A})$ such that $\mathcal{A}$ is the ABox part of some justification for $q, \vec{a}$ in $\mathcal{T} \cup \mathcal{A}$. A* testing base *(TB) for $q, \mathcal{T}$ is a finite set $\mathbf{B}$ of testing units for $q, \mathcal{T}$ such that for each testing unit $\mathcal{A}$ for q and $\mathcal{T}$, there is some $\mathcal{A}' \in \mathbf{B}$ such that $\mathcal{A}'$ is isomorphic to $\mathcal{A}$. A testing base is* minimal *if no two ABoxes in it are isomorphic.*

Consider, as a running example, the following TBox $\mathcal{T}$ stating that everyone taking a maths course is a student and every instance of the relation "takes calculus course" is also an instance of "takes maths course"; consider also the following query q asking for the set of students taking a maths course.

$$\mathcal{T} = \{\exists \mathsf{takesMathCo}.\top \sqsubseteq \mathsf{St}, \mathsf{takesCalcCo} \sqsubseteq \mathsf{takesMathCo}\}$$
$$q = Q_P(x) \leftarrow \mathsf{St}(x) \wedge \mathsf{takesMathCo}(x, y)$$

By Definition 2, the following ABoxes are testing units for $q, \mathcal{T}$, and the set $\mathbf{B} = \{\mathcal{A}_1, \ldots, \mathcal{A}_8\}$ is a minimal TB for $q, \mathcal{T}$:

$$\mathcal{A}_1 = \{\mathsf{takesMathCo}(a, b)\} \quad \mathcal{A}_2 = \{\mathsf{St}(a), \mathsf{takesMathCo}(a, b)\}$$
$$\mathcal{A}_3 = \{\mathsf{takesMathCo}(a, a)\} \quad \mathcal{A}_4 = \{\mathsf{St}(a), \mathsf{takesMathCo}(a, a)\}$$
$$\mathcal{A}_5 = \{\mathsf{takesCalcCo}(a, b)\} \quad \mathcal{A}_6 = \{\mathsf{St}(a), \mathsf{takesCalcCo}(a, b)\}$$
$$\mathcal{A}_7 = \{\mathsf{takesCalcCo}(a, a)\} \quad \mathcal{A}_8 = \{\mathsf{St}(a), \mathsf{takesCalcCo}(a, a)\}$$

As shown in [13], TBs provide the following completeness guarantee for any CQ answering algorithm ans that is sound and faithful: if ans correctly computes the set of certain answers for each ABox in a TB, then it will also compute the set of certain answers for *any* ABox that is consistent with the TBox. In our example, this means that we only need to check whether $\mathsf{ans}(q, \mathcal{T}, \mathcal{A}_i) = \{a\}$ for each $\mathcal{A}_i \in \{\mathcal{A}_1, \ldots, \mathcal{A}_8\}$ in order to determine if ans will compute the set of certain answers of q w.r.t. $\mathcal{T}$ and any ABox that is consistent with $\mathcal{T}$.

3.3 Strict Testing Bases

Intuitively, to check whether each $\mathcal{A}_i$ in our running example is a testing unit, one would need to compute all justifications for q and each certain answer a in

$\mathcal{T} \cup \mathcal{A}_i$, and then check whether $\mathcal{A}_i$ is the ABox part of one of them. This may be infeasible in practice, as we may need to consider all possible subsets of $\mathcal{T}$.

In this paper, we address this issue by investigating the notion of a *strict testing unit*—a minimal ABox that can produce an answer to q w.r.t. $\mathcal{T}$.

Definition 3. *An ABox $\mathcal{A}$ is a* strict testing unit *for a CQ q and TBox $\mathcal{T}$ if $\mathcal{T} \cup \mathcal{A}$ is consistent and there exists a tuple $\vec{a} \in \mathsf{cert}(q, \mathcal{T}, \mathcal{A})$ such that $\vec{a} \notin \mathsf{cert}(q, \mathcal{T}, \mathcal{A}')$ for each $\mathcal{A}' \subset \mathcal{A}$.*

To check whether $\mathcal{A}$ is a strict testing unit, we only need to find a certain answer that is lost when removing any assertion from $\mathcal{A}$. Furthermore, it can be easily shown that each strict testing unit for q and $\mathcal{T}$ is also a testing unit for q and $\mathcal{T}$, and in our running example only the testing units $\mathcal{A}_1, \mathcal{A}_3, \mathcal{A}_5$, and $\mathcal{A}_7$ are strict. The notion of a strict testing unit leads to that of a *strict testing base*.

Definition 4. *A* strict testing base $\mathbf{B}_s$ *for $q, \mathcal{T}$ is a finite set of strict testing units for $q, \mathcal{T}$ such that, for each strict testing unit $\mathcal{A}$ for $q, \mathcal{T}$, there is some $\mathcal{A}' \in \mathbf{B}_s$ such that $\mathcal{A}'$ is isomorphic to $\mathcal{A}$. Finally, a strict testing base is* minimal *if no two ABoxes in it are isomorphic.*

Given any TB $\mathbf{B}$, we can always construct a strict one $\mathbf{B}_s$ by removing from $\mathbf{B}$ the testing units that are not strict, and hence $\mathbf{B}_s$ is likely to be smaller than $\mathbf{B}$ (in our example, $\mathbf{B}_s = \{\mathcal{A}_1, \mathcal{A}_3, \mathcal{A}_5, \mathcal{A}_7\}$ is a strict and minimal TB).

We next present our main result in this section: although strict TBs are smaller than TBs, they provide *exactly the same* completeness guarantees.

Theorem 1. *Let* ans *be a sound and faithful CQ answering algorithm for $\mathcal{L}$. Let q be a CQ, $\mathcal{T}$ an $\mathcal{L}$-TBox and $\mathbf{B}_s$ a strict TB for q, $\mathcal{T}$. The following property ($\Diamond$) holds for any ABox $\mathcal{A}'$ s.t. $\mathcal{T} \cup \mathcal{A}'$ is consistent: If $\mathsf{ans}(q, \mathcal{T}, \mathcal{A}) = \mathsf{cert}(q, \mathcal{T}, \mathcal{A})$ for each $\mathcal{A} \in \mathbf{B}_s$, then $\mathsf{ans}(q, \mathcal{T}, \mathcal{A}') = \mathsf{cert}(q, \mathcal{T}, \mathcal{A}')$.*

Proof. By contradiction, let $\mathsf{ans}(q, \mathcal{T}, \mathcal{A}) = \mathsf{cert}(q, \mathcal{T}, \mathcal{A})$ for each $\mathcal{A} \in \mathbf{B}_s$ and assume there exists $\mathcal{A}'$ s.t. $\mathcal{T} \cup \mathcal{A}'$ is consistent but $\mathsf{ans}(q, \mathcal{T}, \mathcal{A}') \neq \mathsf{cert}(q, \mathcal{T}, \mathcal{A}')$. Since ans is sound, $\mathsf{ans}(q, \mathcal{T}, \mathcal{A}') \neq \mathsf{cert}(q, \mathcal{T}, \mathcal{A}')$ iff $\mathsf{cert}(q, \mathcal{T}, \mathcal{A}') \not\subseteq \mathsf{ans}(q, \mathcal{T}, \mathcal{A}')$. Hence, let $\vec{a} \in \mathsf{cert}(q, \mathcal{T}, \mathcal{A}')$ be s.t. $\vec{a} \notin \mathsf{ans}(q, \mathcal{T}, \mathcal{A}')$. Since $\vec{a} \in \mathsf{cert}(q, \mathcal{T}, \mathcal{A}')$ and $\mathcal{L}$ does not provide for nominals, there is a minimal (w.r.t. set inclusion), non-empty $\mathcal{A}_{min} \subseteq \mathcal{A}'$ s.t. $\vec{a} \in \mathsf{cert}(q, \mathcal{T}, \mathcal{A}_{min})$. But then, $\mathcal{A}_{min}$ is a strict testing unit by Definition 3. Since $\mathbf{B}_s$ is a strict TB, there exists $\mathcal{A}'_{min} \in \mathbf{B}_s$ isomorphic to $\mathcal{A}_{min}$. Finally, since $\mathcal{A}_{min} \subseteq \mathcal{A}'$ and ans is monotonic and invariant under isomorphisms, we have that $\vec{a} \in \mathsf{ans}(q, \mathcal{T}, \mathcal{A}')$, which is a contradiction. □

Thus, given our example $\mathcal{T}$ and q, to check whether a sound and faithful reasoner correctly computes $\mathsf{cert}(q, \mathcal{T}, \mathcal{A})$ for any ABox $\mathcal{A}$, we only need to check whether it returns all the certain answers w.r.t. $\mathcal{A}_1$, $\mathcal{A}_3$, $\mathcal{A}_5$ and $\mathcal{A}_7$.

3.4 Existence and Size of Strict Testing Bases

In [13] we showed that, unfortunately, there exist CQs and ontologies written in rather simple ontology languages for which a TB does not exist, because infinitely

many testing units would be needed. As already discussed, a strict TB exists whenever a TB does. The converse, however, may not hold, and hence our non-existence results from [13] do not transfer directly to strict TBs. The following example shows a TBox and a CQ for which there is a strict TB containing just one ABox with a single assertion, but for which no TB exists.

Example 1. Consider the following TBox and query:

$$\mathcal{T} = \{A \sqsubseteq \exists R.B, \exists R.B \sqsubseteq B\}; \quad q = Q_P(x) \leftarrow A(x) \wedge B(x)$$

The set $\mathbf{B}_s = \{\{A(a)\}\}$ is a strict TB. However, for any value of n, the ABox $\mathcal{A}_n = \{A(a), R(a, b_1), \ldots, R(b_{n-1}, b_n), B(b_n)\}$ is a testing unit (it is the ABox part of a justification J for the certain answer a in $\mathcal{T} \cup \mathcal{A}_n$, whose TBox part is $\mathcal{T}_J = \{\exists R.B \sqsubseteq B\}$), and $\mathcal{A}_i$ and $\mathcal{A}_j$ are non-isomorphic for any $i \neq j$. Thus no TB exists, because from Definition 2 a TB must be a finite set of testing units.

Although a strict TB may exist even if no TB does, it may not be possible in general to guarantee the existence of one. For instance, if we modify $\mathcal{T}$ from Example 1 to be $\mathcal{T} = \{\exists R.B \sqsubseteq B\}$, no strict TB exists for the same reason that no TB does. The proof of Theorem 2 is identical to the one in [13] for TBs.

Theorem 2. *Let $\mathcal{L}$ be $\mathcal{EL}$, or $\mathcal{FL}_0$, or a DL allowing for transitivity axioms. There is a CQ q and a $\mathcal{L}$-TBox $\mathcal{T}$ for which no strict testing base exists.*

In cases when a TB $\mathbf{B}$ does exist (see Section 4), the corresponding strict TB $\mathbf{B}_s$ is likely to be much smaller. A natural question is how small $\mathbf{B}_s$ can be in comparison to $\mathbf{B}$. We next provide an example of an *exponential reduction in size*.

Example 2. Consider the following TBox and query:

$$\mathcal{T} = \{B \sqsubseteq A_i \mid 1 \leq i \leq n\} \cup \{A_1 \sqcap \ldots \sqcap A_n \sqsubseteq C\}; \quad q = Q_P(x) \leftarrow C(x)$$

Let $\mathbf{B}_s$ and $\mathbf{B}$ be as follows, where $\mathcal{A} = \{A_1(a), \ldots, A_n(a)\}$, $\mathcal{B} = \{B(a)\}$, and $\wp(\mathcal{A})$ is the power set of $\mathcal{A}$:

$$\mathbf{B}_s = \{\mathcal{B}, \mathcal{A}, \{C(a)\}\}; \quad \mathbf{B} = \mathbf{B}_s \cup \bigcup_{\mathcal{A}' \in \wp(\mathcal{A}) \setminus \mathcal{A}} \{\mathcal{B} \cup \mathcal{A}'\}$$

The set $\mathbf{B}_s$ with three testing units is a strict and minimal TB for q, $\mathcal{T}$. Also, given any $\mathcal{A}' \subset \mathcal{A}$, we have that $\mathcal{B} \cup \mathcal{A}'$ is a testing unit since it is the ABox of a justification with $\mathcal{T}' = \{B \sqsubseteq A_j \mid 1 \leq j \leq n, A_j(a) \notin \mathcal{A}'\} \cup \{A_1 \sqcap \ldots \sqcap A_n \sqsubseteq C\}$. Therefore, $\mathbf{B}$ is a minimal TB containing $2^n + 1$ testing units.

Although strict TBs can be exponentially smaller than TBs, this is not always the case. The following example shows that an exponential blowup w.r.t. the size of the TBox may not be avoidable when computing strict and minimal TBs.

Example 3. For $n \geq 1$, consider the TBox $\mathcal{T}_n$ consisting of the following axioms for each $0 \leq j < i \leq n$:[2]

[2] A similar TBox was used in [8] for a different purpose.

$$\overline{X}_0 \sqcap \ldots \sqcap \overline{X}_n \sqsubseteq \bot; \qquad X_0 \sqcap \ldots \sqcap X_n \sqsubseteq B;$$
$$\exists R.\overline{X}_0 \sqsubseteq X_0; \qquad \exists R.X_0 \sqsubseteq \overline{X}_0;$$
$$\exists R.(\overline{X}_i \sqcap X_0 \sqcap \ldots \sqcap X_{i-1}) \sqsubseteq X_i; \qquad \exists R.(X_i \sqcap X_0 \sqcap \ldots \sqcap X_{i-1}) \sqsubseteq \overline{X}_i;$$
$$\exists R.(\overline{X}_i \sqcap \overline{X}_j) \sqsubseteq \overline{X}_i; \qquad \exists R.(X_i \sqcap \overline{X}_j) \sqsubseteq X_i.$$

and the query $q = Q_P(x) \leftarrow B(x)$. Intuitively, $\mathcal{T}_n$ implements the incrementation of an n-bit counter along an R-chain. For each $1 \leq k < 2^{n+1}$, let Z_k be of the form $Z_k = \bigsqcap_{0 \leq i \leq n} Y_i$ with $Y_i \in \{X_i, \overline{X}_i\}$ s.t. the binary number obtained by replacing each Y_i in the chain $Y_0 \ldots Y_n$ with 1 if $Y_i = X_i$ and 0 otherwise is precisely the binary encoding of k. Then, for each $2 \leq j < 2^{n+1}$, the following ABox $\mathcal{A}_j$ is a strict testing unit (and is not isomorphic to any $\mathcal{A}_{j'}$ with $j' \neq j$):

$$\mathcal{A}_j = \{R(a_0, a_1), \ldots, R(a_{j-1}, a_j), Z_j(a_j)\}$$

Existence of a strict TB is ensured by the axiom $\overline{X}_0 \sqcap \ldots \sqcap \overline{X}_n \sqsubseteq \bot$, which precludes the computation of an infinite number of (non-isomorphic) strict testing units by "appending" relevant R-chains an arbitrary number of times (recall that a strict testing unit must be consistent with $\mathcal{T}_n$). It can easily be verified that a strict and minimal TB must contain exponentially many testing units w.r.t. n.

3.5 Measuring the Degree of Completeness

In this section, we turn our attention to measuring quantitatively "how complete" a sound and faithful reasoner is for a fixed query q and TBox $\mathcal{T}$, when completeness guarantees are not provided. To this end, we next introduce the notion of *completeness degree*, which in its most general form can be defined as follows.[3]

Definition 5. *Let* ans *be a sound and faithful CQ answering algorithm for* $\mathcal{L}$ *a DL,* q *a CQ,* $\mathcal{T}$ *an* $\mathcal{L}$*-TBox and* $\mathbf{A}$ *a non-empty set of ABoxes such that, for each* $\mathcal{A} \in \mathbf{A}$*,* $\mathcal{T} \cup \mathcal{A}$ *is consistent and* $\mathsf{cert}(q, \mathcal{T}, \mathcal{A}) \neq \emptyset$*. The* completeness degree *δ of* ans *for* q*,* $\mathcal{T}$ *and* $\mathbf{A}$ *is defined as follows (where* $\sharp\mathbf{S}$ *denotes the number of elements in a set* $\mathbf{S}$*):*

$$\delta_{\mathbf{A}}(\mathsf{ans}, q, \mathcal{T}) = \frac{1}{\sharp\mathbf{A}} \times \sum_{\mathcal{A} \in \mathbf{A}} \frac{\sharp\mathsf{ans}(q, \mathcal{T}, \mathcal{A})}{\sharp\mathsf{cert}(q, \mathcal{T}, \mathcal{A})}$$

Therefore, $\delta_{\mathbf{A}}$ represents the proportion of certain answers w.r.t. ABoxes in $\mathbf{A}$ that ans is able to compute correctly. The specific properties of $\delta_{\mathbf{A}}$, however, will obviously depend on the particular set of ABoxes under consideration. Intuitively, in order to obtain a reasonable measure of completeness for $\mathcal{T}$ and q, the set $\mathbf{A}$ should be chosen such that the following basic properties are satisfied:

[3] The notion given here slightly differs from the one in our previous work.

1. If ans misses a certain answer for some (arbitrary) ABox consistent with the TBox, then $\delta_{\mathbf{A}}(\mathsf{ans}, q, \mathcal{T})$ should be smaller than one.
2. If ans correctly computes some certain answer for some (arbitrary) ABox consistent with the TBox, then $\delta_{\mathbf{A}}(\mathsf{ans}, q, \mathcal{T})$ should be larger than zero.
3. If each certain answer computed by ans is also computed by ans', then $\delta_{\mathbf{A}}(\mathsf{ans}', q, \mathcal{T})$ should be at least as large as $\delta_{\mathbf{A}}(\mathsf{ans}, q, \mathcal{T})$.
4. If Property 3 holds and, in addition, there is an (arbitrary) ABox $\mathcal{A}$ consistent with $\mathcal{T}$ for which ans' computes a certain answer that ans fails to compute, then $\delta_{\mathbf{A}}(\mathsf{ans}', q, \mathcal{T})$ should be strictly larger than $\delta_{\mathbf{A}}(\mathsf{ans}, q, \mathcal{T})$.

Consider our running example CQ q, TBox $\mathcal{T}$, TB $\mathbf{B} = \{\mathcal{A}_1, \dots, \mathcal{A}_8\}$ and strict TB $\mathbf{B}_s = \{\mathcal{A}_1, \mathcal{A}_3, \mathcal{A}_5, \mathcal{A}_7\}$. An algorithm a_1 that ignores $\mathcal{T}$ and simply answers q w.r.t. the data would only compute the correct answers for $\mathcal{A}_2$ and $\mathcal{A}_4$; hence, $\delta_{\mathbf{B}}(\mathsf{a}_1, q, \mathcal{T}) = 0.25$, whereas $\delta_{\mathbf{B}_s}(\mathsf{a}_1, q, \mathcal{T}) = 0$. An algorithm a_2 that handles role inclusions but not existential quantification would only compute the correct answers for $\mathcal{A}_2$, $\mathcal{A}_4$, $\mathcal{A}_6$ and $\mathcal{A}_8$; thus, $\delta_{\mathbf{B}}(\mathsf{a}_2, q, \mathcal{T}) = 0.5$, but we again have $\delta_{\mathbf{B}_s}(\mathsf{a}_2, q, \mathcal{T}) = 0$. Finally, a complete algorithm would compute the correct answers for all ABoxes; hence, $\delta_{\mathbf{B}}(\mathsf{a}_3, q, \mathcal{T}) = \delta_{\mathbf{B}_s}(\mathsf{a}_3, q, \mathcal{T}) = 1$, as desired.

Our example suggests that by choosing a (possibly strict) TB, we can guarantee Properties 1 and 3. Indeed, this can be shown in general as a direct consequence of Theorem 1.

Proposition 1. *The following properties hold for* ans *and* ans' *sound and faithful, q a CQ, $\mathcal{T}$ a TBox and $\mathbf{B}_s$ a strict TB for $q, \mathcal{T}$:*

1. *If* $\mathsf{cert}(q, \mathcal{T}, \mathcal{A}) \neq \mathsf{ans}(q, \mathcal{T}, \mathcal{A})$ *for some $\mathcal{A}$ s.t. $\mathcal{T} \cup \mathcal{A}$ is consistent, then* $\delta_{\mathbf{B}_s}(\mathsf{ans}, q, \mathcal{T}) < 1$.
2. *If* $\mathsf{ans}(q, \mathcal{T}, \mathcal{A}) \subseteq \mathsf{ans}'(q, \mathcal{T}, \mathcal{A})$ *for each $\mathcal{A}$,* $\delta_{\mathbf{B}_s}(\mathsf{ans}, q, \mathcal{T}) \leq \delta_{\mathbf{B}_s}(\mathsf{ans}', q, \mathcal{T})$.

Our running example also illustrates an important advantage of using TBs over strict TBs for measuring completeness degrees, namely that Properties 2 and 4 fail if δ is measured in terms of $\mathbf{B}_s$, but hold if δ is measured w.r.t. $\mathbf{B}$. We finally show that Properties 2 and 4 always hold for TBs provided that the relevant CQ answering algorithm is also compact.

Proposition 2. *The following properties hold for* ans *and* ans' *sound, faithful and compact, q a CQ, $\mathcal{T}$ a TBox and $\mathbf{B}$ a TB for $q, \mathcal{T}$.*

1. *If* $\vec{a} \in \mathsf{ans}(q, \mathcal{T}, \mathcal{A})$ *for some tuple $\vec{a}$ and some ABox $\mathcal{A}$ s.t. $\mathcal{T} \cup \mathcal{A}$ is consistent, then* $\delta_{\mathbf{B}}(\mathsf{ans}, q, \mathcal{T}) > 0$.
2. *If* $\mathsf{ans}(q, \mathcal{T}, \mathcal{A}) \subseteq \mathsf{ans}'(q, \mathcal{T}, \mathcal{A})$ *for each ABox $\mathcal{A}$, and there exists $\mathcal{A}'$ and* $\vec{a} \in \mathsf{ans}'(q, \mathcal{T}, \mathcal{A}')$ *s.t.* $\vec{a} \notin \mathsf{ans}(q, \mathcal{T}, \mathcal{A}')$*, then* $\delta_{\mathbf{B}}(\mathsf{ans}, q, \mathcal{T}) < \delta_{\mathbf{B}}(\mathsf{ans}', q, \mathcal{T})$.

Proof. 1. Suppose that such tuple $\vec{a}$ and ABox $\mathcal{A}$ exist. Since ans is sound, $\vec{a} \in \mathsf{cert}(q, \mathcal{T}, \mathcal{A})$. Since ans is also compact, there is a justification $J = \mathcal{T}_J \cup \mathcal{A}_J$ for q, $\vec{a}$ in $\mathcal{T} \cup \mathcal{A}$ such that $\vec{a} \in \mathsf{ans}(q, \mathcal{T}_J, \mathcal{A}_J)$. But then, $\mathcal{A}_J$ is a testing unit by Definition 2. Since $\mathbf{B}$ is a testing base, there exists an ABox $\mathcal{A}' \in \mathbf{B}$ that is isomorphic to $\mathcal{A}_J$ with $\vec{a}'$ the tuple obtained after the corresponding renaming of $\vec{a}$. Finally, since ans is invariant under isomorphisms and monotonic, we clearly have $\vec{a}' \in \mathsf{ans}(q, \mathcal{T}, A')$ and hence $\delta_{\mathbf{B}}(\mathsf{ans}, q, \mathcal{T}) > 0$.

2. By Proposition 1, $\delta_{\mathbf{B}}(\mathsf{ans}, q, \mathcal{T}) \leq \delta_{\mathbf{B}}(\mathsf{ans}', q, \mathcal{T})$. The property then follows from the following statement, which we show next: there exist $\mathcal{A}_{min} \in \mathbf{B}$ and $\vec{b} \in \mathsf{ans}'(q, \mathcal{T}, \mathcal{A}_{min})$ such that $\vec{b} \notin \mathsf{ans}(q, \mathcal{T}, \mathcal{A}_{min})$. Since ans' is compact, there is a justification $J = \mathcal{T}_J \cup \mathcal{A}_J$ for $q, \vec{a}$ in $\mathcal{T} \cup \mathcal{A}'$ s.t. $\vec{a} \in \mathsf{ans}'(q, \mathcal{T}_J, \mathcal{A}_J)$. By definition of a TB, there exists $\mathcal{A}_{min} \in \mathbf{B}$ isomorphic to $\mathcal{A}_J$ with $\vec{b}$ the result of renaming $\vec{a}$ accordingly. By monotonicity and invariance under isomorphisms of ans', $\vec{b} \in \mathsf{ans}'(q, \mathcal{T}, \mathcal{A}_{min})$. But then, $\vec{b} \notin \mathsf{ans}(q, \mathcal{T}, \mathcal{A}_{min})$ since otherwise by monotonicity and invariance under isomorphisms of ans we have $\vec{a} \in \mathsf{ans}(q, \mathcal{T}, \mathcal{A}')$, which is a contradiction. □

4 Computing Strict Testing Bases

In our previous work [13], we identified sufficient conditions for a TB to exist. We showed that it is always possible to construct a TB for $\mathcal{T}, q$ whenever there exists a UCQ rewriting for q and each subset $\mathcal{T}'$ of $\mathcal{T}$. The connection between the existence of UCQ rewritings and of TBs is relevant for practice: on the one hand, UCQ rewritings are guaranteed to exist if $\mathcal{T}$ is expressed in the DLs underpinning the QL profile of OWL 2, and they may also exist even if $\mathcal{T}$ is in other fragments of OWL 2 (such as the $\mathcal{EL}$ profile); on the other hand, there are currently a number of implemented algorithms for computing UCQ rewritings (e.g., those implemented in the systems QuOnto and REQUIEM).

Roughly speaking, the algorithm for computing a TB for $\mathcal{T}$ and q proceeds as follows: first, for each subset $\mathcal{T}'$ of $\mathcal{T}$ it computes a UCQ rewriting $u_{\mathcal{T}'}$; second, for each such $u_{\mathcal{T}'}$ it constructs a fixed set of individuals whose cardinality is bounded by $\mathsf{var}(u_{\mathcal{T}'})$; finally, it computes the required testing units by instantiating each $u_{\mathcal{T}'}$ with a *valid instantiation*—a (maximal) subset of the mappings from the variables of each CQ in $u_{\mathcal{T}'}$ to individuals satisfying certain properties.

This naive algorithm is not practical since it may need to examine an exponential number of subsets of $\mathcal{T}$. This is required to ensure, on the one hand, that a TB exists and, on the other hand, that all relevant testing units are computed via a valid instantiation. For example, the CQ $Q_P(x) \leftarrow A(x)$ is a UCQ rewriting for the query q and TBox $\mathcal{T}$ from Example 1, but no TB exists for $q, \mathcal{T}$. The algorithm from [13] rejects the input $q, \mathcal{T}$ because it additionally considers the subset $\mathcal{T}' = \{\exists R.B \sqsubseteq B\}$ of $\mathcal{T}$ and finds that no UCQ rewriting exists for $q, \mathcal{T}'$.

We next show that a *strict* TB can be computed solely from a UCQ rewriting u for $\mathcal{T}$ and q, and hence computing a UCQ rewriting for each of the (exponentially many) subsets of $\mathcal{T}$ is no longer required. Indeed, we can compute the strict TB $\mathbf{B}_s = \{A(a)\}$ for q and $\mathcal{T}$ from Example 1 by just computing and instantiating the UCQ rewriting $Q_P(x) \leftarrow A(x)$.

We start by recapitulating the notions from [13] of an *instantiation* of a CQ and a *valid instantiation* for a UCQ.

Definition 6. *Let q be a CQ, B_q the body atoms in q, and π a mapping from all variables of q to individuals. The following ABox is an* instantiation *of q:*

$$\mathcal{A}^q_\pi := \{A(\pi(x)) \mid A(x) \in B_q\} \cup \{R(\pi(x), \pi(y)) \mid R(x, y) \in B_q\}$$

Let $u = \{q_1, \ldots, q_n\}$ be a UCQ and assume w.l.o.g. that $\mathsf{var}(q_i) \cap \mathsf{var}(q_j) = \emptyset$ for $i \neq j$. Let $\mathsf{ind} = \{a_1, \ldots, a_m\}$ be a set of individuals s.t. $m = \sharp\mathsf{var}(u)$ and let Π_{q_i} be the set of all mappings from $\mathsf{var}(q_i)$ to ind. A set $\Pi_u = \Pi^u_{q_1} \uplus \ldots \uplus \Pi^u_{q_n}$ with $\Pi^u_{q_i} \subseteq \Pi_{q_i}$ is a valid instantiation *of u if it is a maximal subset of $\Pi_{q_1} \uplus \ldots \uplus \Pi_{q_n}$ with the following property:*

> *$(*)$: for each $q_i, q_j \in u$ and $\pi \in \Pi^u_{q_i}$, there is no $\pi' \in \Pi_{q_j}$ s.t. π and π' map the distinguished variables in q_i and q_j identically and $\mathcal{A}^{q_j}_{\pi'} \subset \mathcal{A}^{q_i}_{\pi}$.*

Intuitively, when instantiating a CQ q in a rewriting u using a valid instantiation, Property $(*)$ from Definition 6 ensures that there is no "smaller" instantiation of a (possibly different) CQ q' from u. In our running example about students and math courses we have that $u = \{q, q_1, q_2, q_3\}$, with q_1, q_2 and q_3 given as follows, is a UCQ rewriting of q and $\mathcal{T}$:

$$q_1 = Q_P(x_1) \leftarrow \mathsf{St}(x_1) \wedge \mathsf{takesCalcCo}(x_1, y_1)$$
$$q_2 = Q_P(x_2) \leftarrow \mathsf{takesMathCo}(x_2, y_2)$$
$$q_3 = Q_P(x_3) \leftarrow \mathsf{takesCalcCo}(x_3, y_3)$$

For $\mathsf{ind} = \{a_i, b_i \mid 0 \leq i \leq 3\}$, we have that $\Pi = \{x_i \mapsto a_i, y_i \mapsto b_i \mid 2 \leq i \leq 3\}$ is a valid instantiation; in contrast, $\Pi' = \{x_1 \mapsto a_1, y_1 \mapsto b_1\}$ is not valid since the mapping $\pi = \{x_3 \mapsto a_1, y_3 \mapsto b_1\}$ leads to a smaller ABox.

Our previous example clearly shows that non-valid instantiations can lead to ABoxes that are not strict testing units. Furthermore, each ABox obtained by instantiating a CQ in a UCQ rewriting using a valid instantiation is indeed a strict testing unit, as shown by the following lemma.

Lemma 1. *Let u be a UCQ rewriting for $\mathcal{T}$ and q, and let Π_u be a valid instantiation. Then, for each $q_i \in u$ and each $\pi \in \Pi^u_{q_i}$ such that $\mathcal{T} \cup \mathcal{A}^{q_i}_{\pi}$ is consistent, we have that $\mathcal{A}^{q_i}_{\pi}$ is a strict testing unit for $\mathcal{T}, q$.*

Proof. Let π map the distinguished variables of q_i to $\vec{a}$. Then, $\vec{a} \in \mathsf{cert}(q_i, \emptyset, \mathcal{A}^{q_i}_{\pi})$. Since $q_i \in u$, soundness of UCQ rewritings implies $\vec{a} \in \mathsf{cert}(q, \mathcal{T}, \mathcal{A}^{q_i}_{\pi})$. To show that $\mathcal{A}^{q_i}_{\pi}$ is a strict testing unit, it suffices to show that $\vec{a} \notin \mathsf{cert}(q, \mathcal{T}, \mathcal{A}^{q_i}_{\pi} \setminus \alpha)$ for each $\alpha \in \mathcal{A}^{q_i}_{\pi}$. By the contrapositive of the completeness property of UCQ rewritings it suffices to show that $\forall q_j \in u$, $\vec{a} \notin \mathsf{cert}(q_j, \emptyset, \mathcal{A}^{q_i}_{\pi} \setminus \alpha)$. But, this is ensured by Property $(*)$ of Definition 6, as we show next.

By contradiction. For some $q_j \in u$ assume that $\vec{a} \in \mathsf{cert}(q_j, \emptyset, \mathcal{A}^{q_i}_{\pi} \setminus \alpha)$.[4] Then, by the semantics of CQ answering there is a mapping σ from the variables in q_j to the individuals in $\mathcal{A}^{q_i}_{\pi} \setminus \alpha$ that maps the distinguished variables of q_j to $\vec{a}$ and such that $\mathcal{A}^{q_j}_{\sigma} \subseteq \mathcal{A}^{q_i}_{\pi} \setminus \alpha$. Hence, $\mathcal{A}^{q_j}_{\sigma} \subset \mathcal{A}^{q_i}_{\pi}$; however, this contradicts the assumption that $\pi \in \Pi^u_{q_i}$, since Property $(*)$ in Definition 6 would fail. □

To compute strict TBs, we need to consider in the worst case all the possible ABoxes that can be obtained by instantiating a given UCQ rewriting using a given valid instantiation, as shown by the following theorem.

[4] note that q_j could be q_i

Algorithm 1 Compute a strict testing base

Algorithm: $\mathsf{tb}(u)$
Input: a UCQ rewriting u for $\mathcal{T}$, q
1 Compute $u' := \mathsf{sub}(\mathsf{cond}(u))$
2 Construct $\mathsf{ind} := \{a_1, \ldots, a_n\}$ for $n = \sharp\mathsf{var}(u')$
3 Initialize $\mathsf{Out} := \emptyset$
4 **For each** $q_i \in u'$
  For each $\pi : \mathsf{var}(q_i) \mapsto \mathsf{ind}$
    If $\mathcal{T} \cup \mathcal{A}_\pi^{q_i}$ is consistent **then**
      $\mathsf{Out} := \mathsf{Out} \cup \{\mathcal{A}_\pi^{q_i}\}$
      For each $q_j \in u'$
        If $\mathsf{Sig}(q_j) \subseteq \mathsf{Sig}(q_i)$ **and** there exists $\pi' : \mathsf{var}(q_j) \mapsto \mathsf{ind}$ s.t. π, π' map distinguished vars. in q_i and q_j identically and $\mathcal{A}_{\pi'}^{q_j} \subset \mathcal{A}_\pi^{q_i}$ **then**
          $\mathsf{Out} := \mathsf{Out} \setminus \{\mathcal{A}_\pi^{q_i}\}$
3 **Return** Out

Theorem 3. *Let u be a UCQ rewriting for $\mathcal{T}, q$ and let Π_u be a valid instantiation. The following set is a strict testig base for $\mathcal{T}$ and q.*

$$\mathbf{B}_s = \{\mathcal{A}_\pi^{q_j} \mid q_j \in u, \pi \in \Pi_{q_j}^u, \mathcal{T} \cup \mathcal{A}_\pi^{q_j} \textit{ consistent}\}$$

Proof. By Lemma 1, $\mathbf{B}_s$ only contains strict testing units. We show that for each strict testing unit $\mathcal{A}$, there exists $q_j \in u$ and a mapping $\pi \in \Pi_{q_j}^u$ s.t. $\mathcal{A}_\pi^{q_j}$ is isomorphic to $\mathcal{A}$ and thus $\mathbf{B}_s$ is a strict TB. $\mathcal{A}$ being a strict testing unit implies that $\mathcal{T} \cup \mathcal{A}$ is consistent and there exists $\vec{a} \in \mathsf{cert}(q, \mathcal{T}, \mathcal{A})$ s.t. $\vec{a} \notin \mathsf{cert}(q, \mathcal{T}, \mathcal{A}')$ for each $\mathcal{A}' \subset \mathcal{A}$. Since $\vec{a} \in \mathsf{cert}(q, \mathcal{T}, \mathcal{A})$ and $\mathcal{T} \cup \mathcal{A}$ is consistent, the completeness property of rewritings implies that $q_j \in u$ exists s.t. $\vec{a} \in \mathsf{cert}(q_j, \emptyset, A)$. Hence, there exists π from $\mathsf{var}(q_j)$ to individuals in $\mathcal{A}$ that maps the distinguished variables in q_j to $\vec{a}$. Since $\mathcal{A}$ is minimal, there is no other π' that maps q_j or any other $q_i \in u$ to a strict subset of $\mathcal{A}$ and s.t. it maps their distinguished variables to $\vec{a}$. Thus, by maximality, there exists $\pi \in \Pi_{q_j}^u$ s.t. $\mathcal{A}_\pi^{q_j}$ is isomorphic to $\mathcal{A}$. □

According to Theorem 3, an algorithm for computing a strict TB for $\mathcal{T}, q$ must use a valid instantiation to compute all testing units. A naive implementation would check Property $(*)$ from Definition 6 by performing a number of ABox containment tests that is exponential in the number of query variables.

We next present a practical algorithm for computing a strict TB. Algorithm 1 takes a UCQ rewriting u for $\mathcal{T}$ and q (computed using any state of the art rewriting algorithm), and implements several optimisations aimed at reducing the number of ABox containment tests needed to check Property $(*)$.

First, as in our practical algorithm from [13], Algorithm 1 uses condensation and subsumption to reduce the size of the input rewriting. This can avoid (possibly exponentially) many tests when checking Property $(*)$. For instance, subsumption would eliminate the queries q and q_1 from the rewriting for our running example, thus discarding each of their instantiations.

Finally, Algorithm 1 only checks the containment of an instantiation of the form $\mathcal{A}_{\pi'}^{q_j}$ in an instantation of the form $\mathcal{A}_\pi^{q_i}$ whenever all the body predicates in

q_j occur also in q_i. For instance, consider the following TBox and CQ:

$$\mathcal{T} = \{C \sqsubseteq \exists R.\top\} \quad q = Q_P(x) \leftarrow R(x,y) \wedge R(y,z)$$

Given $u = \{q, q_1\}$ with $q_1 = Q_P(x) \leftarrow R(x,y) \wedge C(y)$, neither cond nor sub removes any query. Algorithm 1, however, will not perform any test of the form $\mathcal{A}_{\pi'}^{q_1} \subset \mathcal{A}_{\pi}^{q}$ since q_1 mentions the predicate C, which is not mentioned in q.

As shown by the following theorem, none of these optimisations results in a loss of relevant strict testing units, and the output of Algorithm 1 is a strict TB.

Theorem 4. *Algorithm 1 computes a strict TB for q and $\mathcal{T}$.*

Proof. The application of cond and sub on a UCQ rewriting preseves the soundness and completeness properties, and $u' = \mathsf{sub}(\mathsf{cond}(u))$ is also a UCQ rewriting. We show that each $\mathcal{A}_{\pi}^{q_i} \in \mathsf{Out}$ is a strict testing unit. To this end, we show that π belongs to $\Pi_{q_i}^{u'}$ with $\Pi_{u'}$ a valid instantiation of u' w.r.t. ind. This is so unless the following condition holds: there exists $q_j \in u'$ with $\mathsf{Sig}(q_j) \not\subseteq \mathsf{Sig}(q_i)$ and $\pi' : \mathsf{var}(q_j) \mapsto \mathsf{ind}$ s.t. π and π' map the distinguished variables in q_i and q_j identically and $\mathcal{A}_{\pi'}^{q_j} \subset \mathcal{A}_{\pi}^{q_i}$. This condition, however, cannot hold: q_j has an atom X not occurring in q_i and hence for each π' the ABox $\mathcal{A}_{\pi'}^{q_j}$ has an assertion involving X which cannot occur in $\mathcal{A}_{\pi}^{q_i}$. To show that Out is a strict TB, let $\Pi_{u'}$ be a valid instantiation of u' w.r.t. ind. By Theorem 3 we show that for arbitrary $q_i \in u'$ and $\pi \in \Pi_{q_i}^{u'}$, we have $\mathcal{A}_{\pi}^{q_i} \in \mathsf{Out}$. This is the case because Algorithm 1 considers all possible queries q_i, q_j and all possible mappings from variables in those queries to individuals in ind, and it only excludes from Out ABoxes corresponding to instantiations violating Property $(*)$ from Definition 6. □

We conclude by briefly comparing Algorithm 1 with the TB approximation algorithm from our previous work. In [13], we showed that the approximation algorithm produces only testing units, but not necessarily all those needed to obtain a TB. In fact, the approximation algorithm produces only strict testing units, but not necessarily all those needed to obtain a strict TB, and hence it cannot be used to provide completeness guarantees for sound and faithful CQ answering algorithms. Algorithm 1, in contrast, produces strict TBs, and so can be used to provide such guarantees. Furthermore, as we show in the next section, the computation of strict TBs is computationally feasible in practice.

5 Implementation and Evaluation

We have implemented Algorithm 1 in our prototype tool SyGENiA,[5] which uses REQUIEM[6] for the computation of the UCQ rewritings, and used it to evaluate four systems: Sesame 2.3-prl,[7] OWLim 2.9.1,[8] Minerva v1.5,[9] and Jena v2.6.3[10] in each of its three variants (Micro, Mini and Max).

[5] `http://code.google.com/p/sygenia/`
[6] `http://www.comlab.ox.ac.uk/projects/requiem/home.html`
[7] `http://www.openrdf.org/`
[8] `http://www.ontotext.com/owlim/`
[9] `http://www.alphaworks.ibm.com/tech/semanticstk`
[10] `http://jena.sourceforge.net/`

Table 1. Generation times of strict TBs for each LUBM query (in sec.)

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14
1.3	1.1	0.09	6.3	0.25	2.1	0.6	7	2.7	6	0.06	0.1	0.1	0.05

Table 2. LUBM queries for which completeness can be guaranteed.

System	Completeness Guarantee	Completeness w.r.t. LUBM dataset
Jena Max	Q1-Q14	Q1-Q14
OWLim	Q1-Q5, Q7, Q9, Q11-Q14	Q1-Q14
Minerva	Q1-Q4, Q9, Q11, Q14	Q1-Q14
Jena Mini/Micro	Q1-Q3, Q5, Q11, Q13-Q14	Q1-Q5, Q11, Q13-Q14
Sesame	Q1, Q3, Q11, Q14	Q1-Q5, Q11, Q14

We first ran SyGENiA over the LUBM TBox and queries and computed a strict TB for each query—each LUBM query leads to a UCQ rewriting w.r.t. the TBox, and hence a strict TB is guaranteed to exist.[11] Table 1 presents the generation time for each of these strict TBs. We then used these strict TBs to compute the corresponding completeness degrees for each evaluated system.

In contrast to our previous work, Algorithm 1 ensures that each computed collection of datasets is a strict TB, and hence we can provide completeness guarantees in practice. This is illustrated in Table 2 where, for each system, we list the queries for which testing using strict TBs shows that it is complete for any dataset, and the queries for which it is complete w.r.t. the LUBM dataset.

Our results show that Jena Max is the only system that is guaranteed to be complete for all 14 LUBM queries regardless of the dataset—that is, it behaves exactly like a complete OWL reasoner w.r.t. the LUBM queries and TBox. Furthermore, as already noted in our previous work, completeness w.r.t. the LUBM benchmark is no guarantee of completeness in general; for example, OWLim and Minerva are both complete w.r.t. LUBM (and even w.r.t. to the more expressive UOBM benchmark), but for some queries they were found to be incomplete w.r.t. to our datasets. OWLim is, however, guaranteed to be complete for all LUBM queries that do not involve reasoning with existential quantifiers—a feature not supported by the system. Minerva, which uses a DL reasoner to classify the ontology and explicate subsumption between atomic concepts, is still guaranteed to be complete for only 8 queries; this is because our datasets reveal missing answers that depend on subsumptions between *complex concepts* that are not pre-computed by the system. Jena Mini and Micro are guaranteed to be complete for 7 queries. Surprisingly, Jena Mini behaved exactly like Jena Micro, despite the fact that, in theory, Jena Mini can handle a larger fragment of OWL; these differences are, however, not revealed by the structure of the LUBM TBox and queries. Finally, Sesame is only guaranteed to be complete for 4 of the queries.

[11] Since REQUIEM does not currently support individuals in the queries or transitivity in the TBox, we have replaced the individuals in queries by distinguished variables and dispensed with the only transitivity axiom in the LUBM TBox.

Table 3. Completeness degrees for Jena Mini/Micro

Datasets	Q4	Q6	Q7	Q8	Q9	Q10	Q12
LUBM	1	.83	.87	.83	.64	.83	0
SyGENiA	.68	.003	.04	.058	0	.001	.25

Table 4. Completeness analysis on Galen

	Sesame	**OWLim**	**Jena Mini**	**Minerva**
Q1	∼0	.84	∼0	.97
Q2	.07	.83	.07	.96
Q3	.01	.84	∼0	.96
Q4	.01	.77	.01	1

Concerning completeness degrees, the values we have obtained using strict TBs are in line with those from our previous work. However, as discussed in Section 3.5, a completeness degree value smaller than one should be interpreted with caution when using strict TBs, especially in the case of very small values. For instance, consider the values obtained for Jena Mini/Micro given in Table 3. When using strict TBs, the completeness degree for query $Q9$ is 0%, but the system is clearly able to correctly compute certain answers for some ABoxes (e.g., the LUBM dataset). As already discussed, this is because completeness degree measures based on strict TBs fail to satisfy properties 2 and 4 of Proposition 2.

Finally, we have considerd a small version of Galen (an expressive ontology with complex structure used in medical applications) and four queries asking respectively for the instances of the concepts HaemoglobinConcentrationProcedure, PlateletCountProcedure, LymphocyteCountProcedure, and HollowStructure. Each of these queries has a UCQ rewriting that can be computed using REQUIEM. Thus, a strict TB exists for each of them and can be computed in times ranging from 2 seconds to 1 minute. Our results are summarised in Table 4.

We could not run Jena Max since Galen makes heavy use of existential restrictions, which (according to the Jena documentation) might cause problems. Among the other systems, Minerva exhibited the best behavior: it was the only one for which completeness could be guaranteed for at least one query, and it exhibited a high completeness degree for the remaining three queries; this is because Minerva pre-computes many subsumption relationships between atomic concepts that depend on existential restrictions, which most other systems do not handle. Jena Mini and Sesame were surprisingly incomplete, although as already discussed, values close to zero should be interpreted with caution.

6 Conclusion and Future Work

In this paper we have extended in several important ways our prior work on completeness evaluation of Semantic Web reasoners. Most importantly, we have introduced the notion of a strict testing base, studied its formal properties,

and shown that it can be used to identify circumstances in which completeness guarantees can be provided for reasoners that are incomplete in general. Finally, we have proposed a practical algorithm for the generation of strict testing bases, implemented it in the SyGENiA tool, and used SyGENiA to evaluate several incomplete reasoners, using both the LUBM benchmark, and the Galen ontology.

Our results suggest not only that strict testing bases are relatively easy to compute in practice, but also that completeness guarantees can often be provided for realistic ontologies and queries. The main limitation of strict testing bases is that the associated completeness degree fails to satisfy certain desirable properties. An interesting problem for future work is to try to design a practical algorithm that can be used to provide an accurate measure of completeness degree.

Acknowledgments Supported by the EU project SEALS (FP7-ICT-238975). B. Cuenca Grau is supported by a Royal Society University Research Fellowship.

References

1. Baader, F., McGuinness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook: Theory, implementation and applications. Cambridge Uni. Press (2002)
2. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. J. of Automated Reasoning 39(3), 385–429 (2007)
3. Glimm, B., Horrocks, I., Lutz, C., Sattler, U.: Conjunctive query answering for the description logic $\mathcal{SHIQ}$. In: Proc. of IJCAI-07 (2007)
4. Guo, Y., Pan, Z., Heflin, J.: LUBM: A Benchmark for OWL Knowledge Base Systems. Journal of Web Semantics 3(2), 158–182 (2005)
5. Hitzler, P., Vrandecic, D.: Resolution-based approximate reasoning for OWL DL. In: Proc. of ISWC-05. pp. 383–397 (2005)
6. Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E.: Finding all justifications of OWL DL entailments. In: Proc. of ISWC-07. pp. 267–280 (2007)
7. Lutz, C., Toman, D., Wolter, F.: Conjunctive query answering in the description logic $\mathcal{EL}$ using a relational database system. In: Proc. of IJCAI-09 (2009)
8. Lutz, C., Wolter, F.: Conservative extensions in the lightweight description logic $\mathcal{EL}$. In: In Proc. of CADE-07. pp. 84–99 (2007)
9. Ortiz, M., Calvanese, D., Eiter, T.: Characterizing data complexity for conjunctive query answering in expressive description logics. In: Proc. of AAAI-06 (2006)
10. Pan, J.Z., Thomas, E.: Approximating OWL-DL Ontologies. In: Proc. of AAAI-07. pp. 1434–1439 (2007)
11. Pérez-Urbina, H., Horrocks, I., Motik, B.: Efficient query answering for OWL 2. In: Proc. of ISWC 09 (2009)
12. Poggi, A., Lembo, D., Calvanese, D., Giacomo, G.D., Lenzerini, M., Rosati, R.: Linking data to ontologies. J. Data Semantics 10, 133–173 (2008)
13. Stoilos, G., Cuenca Grau, B., Horrocks, I.: How incomplete is your semantic web reasoner? In: Proc. of AAAI-10 (2010)